CISC 322/326 ASSIGNMENT 1: REPORT

GNUstep: Conceptual Architecture Analysis

February 14, 2025

Group 10
Chi Ma (Presenter)
Nick He
Zhiming Jin
Tiantian Sang (Presenter)
Dunyi Xie
Lixing Yang (Leader)

Table of Contents

1 Abstract

Our report presents a comprehensive analysis of GNUstep's architecture and design, highlighting its evolution, key components, and the lessons learned during its development. GNUstep is an open-source implementation of the OpenStep specification and serves as a robust framework for cross-platform application development. Drawing inspiration from NeXTSTEP and Cocoa, GNUstep employs a modular, object-oriented architecture that enables developers to build applications with consistent behavior across GNU/Linux, Windows, and BSD systems.

A central focus of the report is GNUstep's layered architectural style, which enhances maintainability and flexibility by enforcing a strict separation of concerns. The report also details the derivation process of GNUstep, which began with an effort to port HippoDraw from NeXTSTEP using a reimplementation of the NeXTSTEP object layer, libobjcX. Key design patterns play a significant role in GNUstep's architecture. The Model-View-Controller (MVC) pattern is extensively utilized to separate data management (Model, thereby promoting a clear division of responsibilities and facilitating independent development of each component. Concurrency is another critical aspect addressed in the report. GNUstep supports multi-threading via NSThread, allowing background tasks such as file processing and image manipulation to execute without compromising the responsiveness of the user interface. Additionally, the report explores practical use cases and discusses the division of responsibilities among GNUstep's modules. It highlights the importance of stable, well-defined interfaces and cross-team collaboration.

In summary, the report finds that GNUstep's layered, object-oriented architecture, combined with robust concurrency support and adherence to proven design patterns, offers a powerful and flexible foundation for cross-platform development. The lessons learned underscore the significance of modularity, clear separation of concerns, and effective synchronization in building scalable, maintainable, and high-performance software systems.

2 Introduction and Overview

2.1 Purpose of the Report

The primary purpose of this report is to analyze the architectural style, design patterns, and developmental processes that have contributed to GNUstep's longevity and versatility. In doing so, the report not only examines the technical foundations of GNUstep but also discusses its practical implications for software development, including concurrency management, control and data flow, and the division of responsibilities across its modules.

2.2 Organization of the Report

The report is organized into the following sections:

1. Abstract: A concise summary of the report's key points and findings.
2. Introduction and Overview: This section sets the stage by purpose of the report, and outlining the main topics covered in the report.
3. Derivation Process: An exploration of how GNUstep was developed and its architecture style and design pattern.

4. Interacting Parts: A detailed analysis of GNUstep's core components—such as the Foundation library, GUI library, and Back library—and how they interact within a layered architecture.
5. Software Development Stages: A chronological review of GNUstep's development, from its inception in the early 1990s to its current state, highlighting key milestones.
6. Concurrency: An examination of GNUstep's approaches to handling concurrent tasks, including multi-threading support through NSThread and event-driven processing via NSRunLoop.
7. Control and Data Flow: A discussion on how user interactions and data processing are managed within GNUstep, ensuring smooth and efficient application performance.
8. Use Cases: Practical scenarios demonstrating GNUstep's application in real-world contexts, such as collaborative multi-device editing and conflict resolution.
9. Division of Responsibilities: An analysis of how the various modules and development teams collaborate within the GNUstep ecosystem, ensuring modularity and stable interfaces.
10. Lessons Learned: Key insights and takeaways from GNUstep's development process, including the importance of cross-platform compatibility, modular design, and effective concurrency management.
11. References: A compilation of sources and documentation that underpin the report's analysis and conclusions.

2.3 Salient Conclusion

The analysis conducted in this report leads to several salient conclusions:

- Layered Architecture and Modularity: GNUstep's design, based on a layered architecture, provides clear separation of concerns—dividing the system into foundational, GUI, and rendering layers. This separation not only enhances maintainability and modularity but also simplifies cross-platform deployment.
- Object-Oriented Design and Design Patterns: Central to GNUstep's architecture is its object-oriented foundation, which encourages code reusability and encapsulation. The extensive use of design patterns such as Model-View-Controller (MVC), Delegation, Factory, and Adapter has facilitated a robust and scalable development environment.
- Cross-Platform Compatibility: By abstracting platform-specific details within the Foundation and Back layers, GNUstep allows developers to write applications that run on multiple operating systems with minimal modifications, a critical advantage in today's diverse computing environments.
- Effective Concurrency Management: GNUstep leverages multi-threading through NSThread and synchronizes shared resources using NSLock and @synchronized, while NSRunLoop ensures that GUI events are processed orderly. These mechanisms are vital for maintaining responsiveness, especially in applications that require simultaneous background processing.
- Practical Use Cases: Real-world scenarios, such as collaborative editing and document export, demonstrate the practical application of GNUstep's architecture. These use cases underline how the framework supports complex workflows and ensures that updates and changes are efficiently propagated across devices.

- Division of Responsibilities: The report highlights how a well-defined separation between core libraries and auxiliary tools—such as GNUstep Make, Gorm, and ProjectCenter—enables different teams to work independently on various modules. This approach reduces coupling, improves collaboration, and streamlines the overall development process.

In summary, the report underscores that GNUstep's innovative and modular architecture, bolstered by its object-oriented design and effective concurrency management, offers a powerful foundation for cross-platform application development. The insights gleaned from this study not only highlight GNUstep's technical strengths but also serve as valuable lessons in building scalable, maintainable, and responsive software systems.

3 Derivation Process

At the beginning of our investigation, we studied the project's documentation and wiki, trying to understand its conceptual structure. As a brief introduction, GNUstep is an open source implementation of the OpenStep specification that provides a framework for developing cross-platform applications. Originally inspired by NeXTSTEP, GNUstep allows developers to create applications across different operating systems using a consistent API. In addition, it provides a complete development environment that supports Objective-C and integrates with various graphical user interfaces. GNUstep's modular architecture ensures compatibility with modern operating systems, including Linux, Windows, and BSD, making it a viable solution for cross-platform software development.

3.1 Architectural Style

Based on our initial understanding of GNUstep, we studied its architectural style and proposed several hypotheses. First, we determined that a layered architecture was the primary design choice. Several reasons supported this hypothesis.

Separation of Concerns: GNUstep adopts a layered approach, dividing the system into a presentation layer, an application logic layer, and a data access layer. This separation enhances maintainability and modularity, allowing developers to modify one layer without affecting others.

Encapsulation of Components: The core functionalities of GNUstep are divided into well-defined modules such as libs-base, which provides fundamental utilities, and libs-gui, responsible for graphical user interface elements. This modular structure ensures code reusability and ease of maintenance.

Platform Independence**:** GNUstep follows a structured framework that abstracts platform-specific details. By using a standardized API, applications built with GNUstep can run on multiple operating systems without significant modifications.

Besides this, another important architectural style of GNUstep is object-oriented architecture. GNUstep is fundamentally built on object-oriented principles, which allow for a high degree of modularity and code reusability. Components are encapsulated within objects, each responsible for specific functionalities, making it easier to extend and maintain the system

over time. By following an object-oriented approach, GNUstep ensures that different modules can interact through well-defined interfaces while remaining loosely coupled. This design improves extensibility, as new features can be added without significantly changing the existing structure.

3.2 Design Patterns

One of the most prominent patterns used in GNUstep is the Model-View-Controller (MVC) pattern, which structures applications by separating data management (Model), user interface (View), and interaction logic (Controller). This separation enhances flexibility, allowing the UI to change without affecting business logic.

To handle interactions between objects efficiently, GNUstep relies on the Delegation pattern, commonly found in "NSApplicationDelegate" and "NSTableViewDelegate". Instead of requiring extensive subclassing, delegation allows objects to assign responsibilities to other classes, improving modularity. Additionally, GNUstep employs the Factory pattern to create objects dynamically.

For system compatibility and flexibility, GNUstep incorporates the Adapter pattern, which bridges Objective-C APIs with lower-level system components like X11 backends.

Overall, GNUstep's architecture is deeply influenced by object-oriented and component-based design patterns, ensuring it remains a scalable and maintainable framework for cross-platform GUI and application development.

4. Interacting Parts

GNUstep is fundamentally a layered system, where each layer builds upon lower layers and provides services to layers above.

4.1 Base (Foundation) Library

At the heart of GNUstep is the Foundation library (commonly referred to as GNUstep-base), which constitutes a powerful non-graphical layer of classes and functionalities. It encapsulates everything from data structures (strings, arrays, dictionaries) to file I/O, networking, and concurrency. This tier can be compared to the "Common code" in other systems: it is the dependable workhorse that higher-level modules rely on for day-to-day tasks. Foundation's primary strength lies in how it abstractly interacts with the operating system, offering a uniform API regardless of whether the user is on Linux, a BSD system, or even Windows. Thus, when developers build logic—be it loading configuration files or processing network traffic—Foundation provides a single, consistent interface, insulating user code from many OS-specific details.

4.2 GUI Library

Sitting on top of Foundation is the GUI library (GNUstep-gui), akin to Cocoa's AppKit. This module focuses on user-facing features: drawing windows, managing button clicks, handling text fields, and more. By design, the GUI library delegates its lower-level drawing commands to a special subsystem known as the Back (or GNUstep-back). The GUI library interacts richly with Foundation: from retrieving localized strings and reading preference files to spawning worker

threads that handle background tasks. Consequently, a developer working with GNUstep's GUI layer can concentrate on how the interface behaves—responding to user gestures, updating labels, or arranging sub views—without worrying about how bits are drawn on the screen or how window events are sourced from the underlying OS. Those specifics are all managed by the tier below.

4.3 Back (Rendering) Library

Meanwhile, the Back layer works like a platform abstraction for rendering and event handling. Conceptually, it is like the "backends" layer in other multi-platform projects. Where the GUI library thinks in terms of "draw a rectangle here" or "show a menu at this location," the Back translates those broad requests into the appropriate native API calls—be it X11 (on Unix-like systems), Windows GDI, or an OpenGL/Cairo-based driver. In the same breath, it captures keyboard and mouse events from the operating system, forwarding them up to the GUI library, which in turn dispatches those events to the correct buttons, windows, or text fields. By maintaining a strict boundary between "what to draw" (the GUI library) and "how to draw it" (the Back layer), GNUstep can be readily ported to new platforms simply by writing or adapting a new back-end, all without altering the higher-level user interface code.

4.4 Developer Tools and Auxiliary Components

This broad category includes items like GNUstep Make, Gorm, and ProjectCenter. They do not form a strict "runtime" layer but are crucial for building and managing GNUstep applications.

4.4.1GNUstep Make

To orchestrate the building and packaging of these components, GNUstep Make (gnustep-make) plays a crucial, albeit behind-the-scenes, role. It is not a direct runtime subsystem like Foundation, GUI, or Back. Instead, GNUstep Make ensures consistent directory structures, compilation flags, and linker settings across different platforms. A developer writing an application simply references the GNUstep Makefiles, and the system automatically knows which libraries to pull in, how to handle Objective-C flags, and where to bundle resources. This means that even though the developer might produce an .app bundle for a Linux-based environment today, they can usually rebuild the same code on Windows tomorrow—only needing the corresponding backend library—without rewriting all the build logic.

4.4.2 Gorm and ProjectCenter

Layered atop all of this, or woven throughout, are additional developer tools such as Gorm (an Interface Builder equivalent) and ProjectCenter (a lightweight IDE). These tools are themselves GNUstep applications built with Foundation, GUI, and the Back, demonstrating the framework's self-referential design. Gorm communicates with the GUI library to present a drag-and-drop interface for designing windows, buttons, or menus; it then serializes these layouts into .gorm files. During runtime, an application loads these user interface definitions, hooking them into its Objective-C classes seamlessly. This approach highlights how the frameworks interact: Gorm uses the same libraries as any developer's program, guaranteeing that what you see while designing an interface is precisely what you get at runtime.

5 Software Development Stages

GNUstep's story began in Stanford Linear Accelerator Center when Paul Kunz and his team aimed to port HippoDraw from NeXTSTEP to a different platform. They chose to rewrite the NeXTSTEP object layer that the application relied on instead of completely rewriting HippoDraw and only using the application design. Which is the first version of libobjcX. By this

method, they could port HippoDraw to Unix systems running the X Window System without altering any lines of the application's source code. After OpenStep specification was published in 1994, they decided to develop a new version of objcX that would comply with the new APIs. This software eventually became "GNUstep." as public known.

5.1 Early Development (1993-1994)

Paul Kunz and his team first focus on developing a free software version of the NeXTSTEP, finally created libobjcX. *"NeXT Computer Inc, and Sun Microsystems Inc. teamed up in late 1993 to push a free object layer API based on the NeXTSTEP object system. This agreement evolved into the OpenStep specification which was published by NeXT in a first draft back in summer 1994 (https://www.gnustep.org/information/openstep.html)."*

5.2 OpenStep Implementation (1994-1995)

After the release of the OpenStep specification, GNUstep developers worked on creating a new implementation that adhered to the OpenStep APIs.

5.3 GNUstep Base and GUI (1995-2000)

In this period, the development of the Foundation and Application Kit libraries published, which provide the core functionality for GUI applications.

5.4 Stabilization and Expansion (2000-2010)

GNUstep continued to stabilize its core libraries and expand its capabilities, including support for additional platforms and improved compatibility with Apple's Cocoa APIs.

5.5 Modern GNUstep (2010-Present)

Working on maintaining compatibility with modern Cocoa APIs, improving performance, and expanding the ecosystem with Gorm (Interface Builder) and ProjectCenter (Project Builder/Xcode) and so on.

6 Concurrency

6.1 Introduction - Overview of Concurrency in GNUstep

Concurrency is important in software because it helps systems run multiple tasks at the same time. This makes software faster and more responsive. In GNUstep, concurrency is needed for two main reasons.

First, responsiveness. If a program only runs one task at a time, the user interface (UI) might freeze while waiting for something to finish. For example, if the system is downloading a file or running a complex calculation, the user should still be able to click buttons or scroll through the interface. Without concurrency, the whole system might become unresponsive.

Second, background tasks. Many applications need to handle multiple operations at once, like processing data, loading images, or waiting for user input. Concurrency allows GNUstep programs to perform these tasks in the background without blocking the main application.

6.2 Concurrency Mechanisms in GNUstep

GNUstep provides a few ways to handle concurrency. The two most important ones are multi-threading and event-driven concurrency.

6.3 Multi-threading Support

GNUstep supports multi-threading using NSThread. Developers can create multiple threads, each running independently. This is useful for handling multiple tasks at once, such as processing user input while loading a file in the background. However, when multiple threads access the same data, problems like race conditions and deadlocks can happen. A race condition occurs when two threads try to change the same data at the same time, making the final result unpredictable. Deadlocks happen when two threads wait for each other to finish, causing the program to freeze. To prevent these issues, GNUstep provides NSLock and @synchronized. NSLock allows only one thread to access shared data at a time, preventing race conditions. @synchronized ensures that only one thread can run a critical section of code at any moment. To avoid deadlocks, developers should always acquire locks in the same order to prevent threads from waiting on each other indefinitely.

6.4 Event-Driven Concurrency in GUI

GUI updates should not be handled by multiple threads at the same time. If different threads try to update the interface together, the program might crash or behave unpredictably. Instead of using multiple threads, GNUstep uses NSRunLoop to process events one at a time, ensuring UI consistency. For example, when a user clicks a button, NSRunLoop adds the event to a queue and processes it in order, making sure UI updates do not happen at the same time from different threads. This prevents visual glitches and ensures smooth application response. By handling GUI concurrency through NSRunLoop, GNUstep reduces the risk of UI-related race conditions and ensures stable interface behavior.

6.5 Conclusions

Concurrency is necessary to make applications responsive and efficient, but it also introduces complexity. GNUstep provides tools like NSThread for background tasks and NSRunLoop for handling events in GUI applications. While these features improve performance, they also require careful management to avoid issues like race conditions and deadlocks.

Instead of relying only on multi-threading, developers should use the right approach for different tasks. Synchronization tools like NSLock help manage shared data, while event-driven models like NSRunLoop keep the UI stable. A well-balanced use of concurrency makes GNUstep applications run smoothly without unnecessary overhead.

7 Control and Data Flow

In GNUstep, control and data flow begin when a user interacts with the GUI. The input is first handled by the front end, which processes actions like button clicks and text input. This interaction is then passed to the application logic layer, where controllers interpret the commands and execute necessary operations.

The data is then processed by the middle end, which manages core functionalities like file handling, state updates, and cross-platform services. This layer abstracts system-specific details

and ensures smooth communication between the front and back ends. If required, the request is forwarded to the back end, which interacts with system resources such as databases and file systems to retrieve or store information. Once processed, the data flows back through the middle end to update the user interface.

GNUstep follows an event-driven model, ensuring tasks are executed asynchronously to maintain responsiveness. It also employs cooperative multitasking, where processes yield control when necessary to avoid blocking system operations.

This structured flow ensures efficient management of user interactions, data processing, and system execution while maintaining modularity and cross-platform support. The following diagram provides a visual representation of the process:

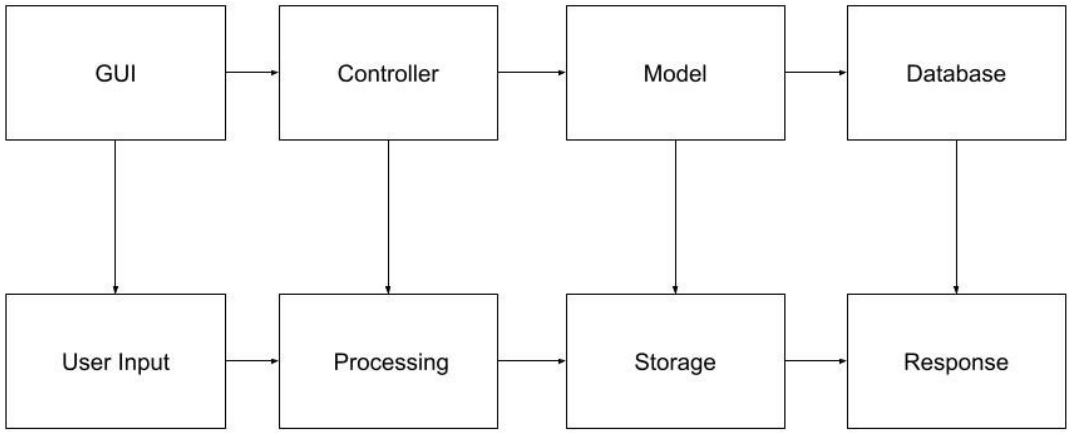


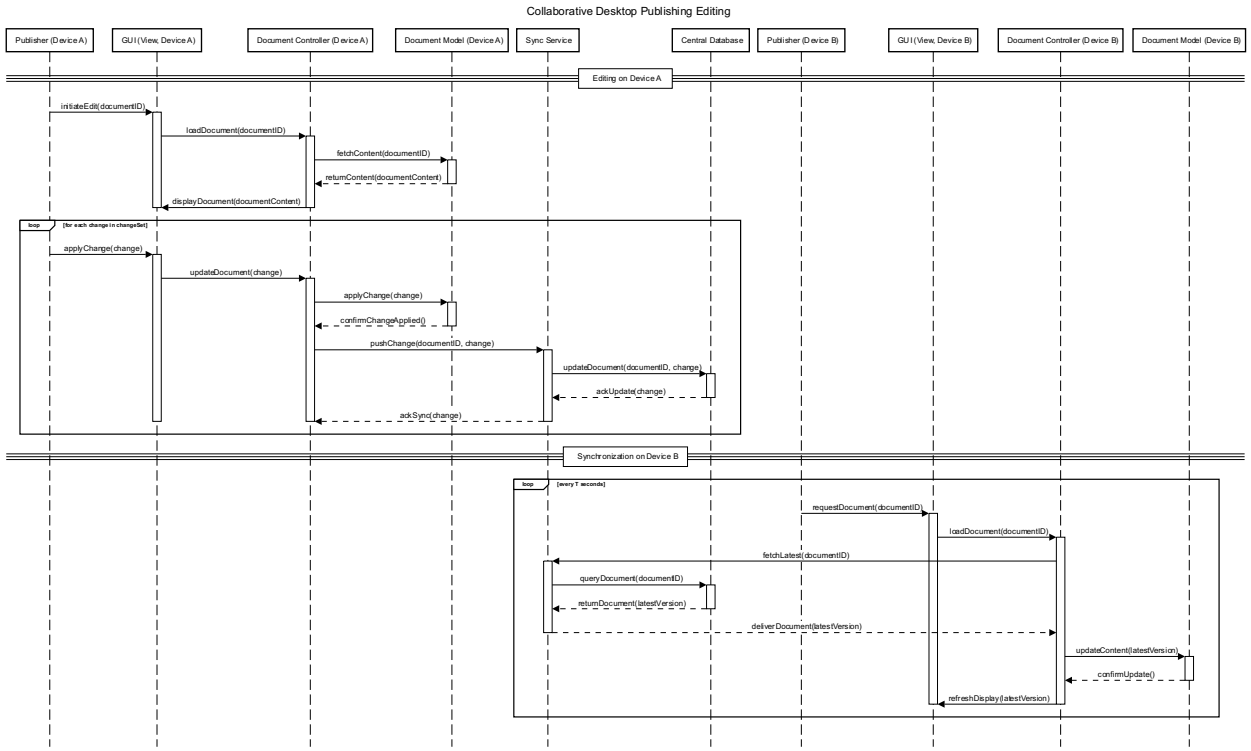
Figure: Box Arrow Diagram of Data Flow

8 Use Cases

GNUstep is a free, open-source development environment that implements the OpenStep and Cocoa API specifications, built around a robust, modular, and object-oriented architecture. At its core, GNUstep leverages the Model-View-Controller (MVC) paradigm to ensure a clean separation of concerns—its Foundation Kit provides essential data structures and utilities, while the Application Kit (AppKit) handles user interface elements. This architecture enables developers to build highly reusable and maintainable applications that run consistently across multiple platforms, including GNU/Linux, Windows, and BSD systems. Additionally, GNUstep's design emphasizes portability and extensibility, allowing developers to integrate new functionalities via plugins and additional libraries while preserving the elegance and simplicity of Objective-C programming.

8.1 Use Case 1: Multi-Device Application

This UML sequence diagram illustrates a multi-device, collaborative desktop publishing workflow. On Device A, the user (Publisher_A) initiates and applies changes to a document via the GUI, Document Controller, and Document Model. Each change is then pushed to a central Sync Service, which updates a shared database. Meanwhile, on Device B, another user (Publisher_B) periodically requests the latest document version. The Sync Service retrieves updates from the central database, and the Document Controller on Device B merges them into its local Document Model, refreshing the GUI to display the most recent content. By showing both an editing loop on Device A and a synchronization loop on Device B, this diagram emphasizes how changes are captured, stored, and distributed across devices in near real time, ensuring consistent, up-to-date document states for all collaborators.



Figure：Multi-Device Application publishing workflow

8.2 Use Case 2: Collaborative Documents

This sequence diagram illustrates the collaborative document editing process with multiple loops and branching logic to handle real-time edits and conflict resolution. The process begins when a user opens a document, and the system checks if the document exists on the server. If it does, the current version is retrieved from the database and displayed in the GUI. Once editing begins, the first loop continuously processes user edits, where each edit is sent to the controller for validation and synchronization with the server. The server updates the database and broadcasts changes to other clients. Within this loop, an alt block handles conflict detection: if a conflict is detected, the system attempts to merge the changes and notifies the user of the result. If no conflict occurs, the updates are broadcast seamlessly to other users. The second loop periodically triggers server updates, where the server broadcasts document changes to connected clients or checks for unresolved conflicts, ensuring that all users are kept in sync with the latest version of

the document. This flow ensures a robust, real-time collaboration experience while maintaining data integrity.

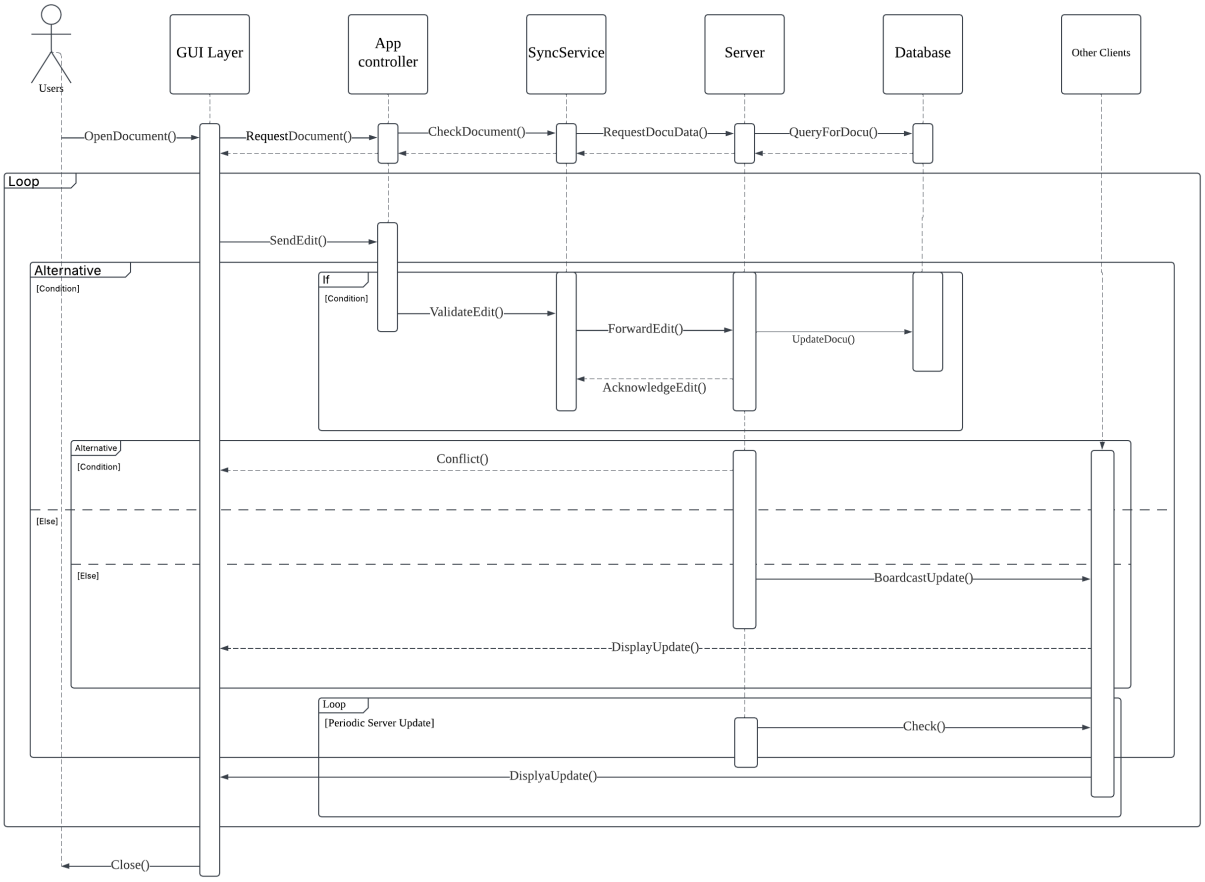


Figure: The Collaborative Document Editing Process

9 Division of Responsibilities

In GNUstep, the division of responsibilities among developers is significantly influenced by the dependency structure of different modules.

9.1 Modules have no dependencies

The core and foundation level libraries ("Libs-base" and "libs-codebase") provide fundamental functionality such as data structures, file operations, memory management, networking, and utilities that higher-level modules (GUI framework and Gorm) depend upon. Typically, they rely only on the standard C library or minimal OS interfaces. They can be worked independently by a team, provided they maintain a stable, well-documented API.

File system and data processing modules load, store, or process user documents, configuration files, or network data. Since they do not usually require knowledge of the GUI, these modules

can be developed independently and in parallel, with focus only on read/write protocols or path management.

9.2 Modules with dependencies

User interface (libs-gui) and Rendering Back End (libs back) depend on the core module, while also depend on each other. Libs-gui handles window management, UI controls, and event distribution, while libs-back handles low-level rendering, window system interaction, and cross-platform support. These two components are tightly connected, as new GUI features may require backend rendering updates, and backend optimizations may necessitate changes in GUI interfaces. When libs-gui modifies event handling processes, libs-back must adapt accordingly. Similarly, if the backend introduces hardware-accelerated rendering, the GUI must adjust its widget drawing logic to accommodate these changes. Therefore, the development of these modules should be closely coordinated with the core team and themselves.

Gorm is responsible for visually designing interfaces — creating windows, menus, and layouts — then saving them to nib/gorm files that the GUI layer will load. Because Gorm must accurately represent the properties and structure of each control available in the GUI, any changes in the GUI's controls (such as new properties) updates Gorm's code. Therefore, Gorm developers rely on stable widget definitions and archiving protocols within libs-gui. Ideally, the GUI framework reaches a certain level of maturity before major Gorm development proceeds.

In summary, independent modules can be developed early and in parallel to ensure stable APIs. For highly dependent modules, their dependencies should be clearly defined, and their development should build on top of core modules.

9.3 Cross-team collaboration

When multiple teams develop different modules, detailed design documents and clearly defined interfaces are crucial. Each library has a well-defined public API. By establishing clear API protocols, teams can ensure low coupling and effective collaboration across different components.

Each component can have its own repository or branch (such as gnustep-base). Teams work on separate branches for their components and merge changes into the main branch after code reviews and testing.

Regular syncs and communication are necessary to ensure that all members stay informed about project progress and can plan around changes in dependent modules. Weekly or bi-weekly Zoom calls among the UI, backend, and Gorm teams help ensure that UI changes and new features in Gorm remain aligned. However, the libs-base team typically coordinates with other teams only when fundamental changes are planned.

Finally, a comprehensive automated testing framework is essential to prevent changes in one component from breaking others. For example, unit tests for libs-base ensure its reliability, while integration tests verify the compatibility of libs-gui with the latest version of libs-base and libs-corebase.

10 Lessons Learned

10.1 From Derivation Process

One important lesson we learned was the importance of cross-platform compatibility in software development. GNUstep's ability to abstract platform-specific details enables it to run on multiple operating systems, a key factor in modern software design. This highlights the importance of designing an architecture that supports flexibility and longevity.

10.2 From Interacting Parts

GNUstep is organized into four main components that form a cohesive, layered ecosystem. Foundation (gnustep-base) serves as the non-UI core, handling collections, file I/O, networking, and other OS abstractions. Building on top of Foundation is the GUI (gnustep-gui), which provides classes and interfaces for user-facing elements, but defers low-level rendering and event handling to the Back (gnustep-back) layer. That Back layer, in turn, communicates directly with platform-specific APIs—like X11 or Windows GDI—ensuring portability across different systems. Finally, GNUstep Make & Developer Tools simplify building and maintenance: GNUstep Make enforces a unified project structure and compilation process, while tools like Gorm (interface builder) and ProjectCenter (IDE) further streamline application development.

10.3 From Concurrency

Understanding concurrency in GNUstep helped us see why managing multiple tasks at the same time is important but also challenging. Multi-threading using NSThread allows different parts of a program to run separately, making applications faster and more responsive. However, we also learned that when multiple threads share the same data, problems like race conditions and deadlocks can happen. GNUstep provides NSLock and @synchronized to solve these issues, but developers still need to be careful about how they manage threads. For GUI applications, we saw that NSRunLoop is used instead of multi-threading to handle events in order, preventing crashes and unexpected UI behavior. Another lesson we learned is that too many threads do not always mean better performance. From the readings, we saw that software architecture should balance concurrency and efficiency, making sure that multiple tasks do not slow down the system instead of improving it.

10.4 From Control and Data Flow

Analyzing control and data flow highlights the importance of a well-defined middleware layer. The middle-end in GNUstep acts as a bridge between the front-end and back-end, ensuring smooth communication between different components. This emphasizes the value of designing a middle-end that efficiently handles logic and abstractions.

10.5 From Division of Responsibility

GNUstep's development follows a modular approach, enabling independence between core libraries and file system management. However, certain dependent modules, such UI interactions and backend rendering, require closer coordination

To maintain effective cross-team collaboration, GNUstep relies on well-defined interfaces, branch-based code management, and automated testing. By ensuring that each module has a stable API, teams can work independently while integrating their changes. Regular syncs and continuous integration testing help set up task division and ensure smooth project development.

11 Reference

1. GNUstep MediaWiki. Main page. Retrieved from https://mediawiki.gnustep.org/index.php/Main_Page
2. Clemson University. GNUstep manual. Retrieved from http://andrewd.ces.clemson.edu/courses/cpsc102/notes/GNUStep-manual.pdf
3. GNUstep. GUI Reference. Retrieved from https://www.gnustep.org/resources/documentation/Developer/Gui/Reference/index.html
4. GNUstep. Base Library Reference. Retrieved from https://www.gnustep.org/resources/documentation/Developer/Base/Reference/index.html
5. GNUstep. GNUstep Developer Documentation. Retrieved from https://www.gnustep.org/developers/documentation.html
6. GNUstep. GNUstep Base Library API Reference. Retrieved from https://www.gnustep.org/resources/documentation/Developer/Base/Reference/Base.html
7. GNUstep. NSThread Class Reference. Retrieved from https://www.gnustep.org/resources/documentation/Developer/Base/Reference/NSThread.html
8. GNUstep. NSLock Class Reference. Retrieved from https://www.gnustep.org/resources/documentation/Developer/Base/Reference/NSLock.html
9. GNUstepWiki. Foundation. Retrieved from https://mediawiki.gnustep.org/index.php/Foundation